ML 01: LINEAR REGRESSION

Machine Learning by Andrew Ng, *Coursera*

What is Machine Learning?

> Field of study that gies computers the abiliry to learn without being explicitly programmed. (1959, Arthur Samuel)
>
> **Well-posed Learning Problem:** A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E (1998, Tom Michell)

체크메이트를 예로 들면, 수천번의 체스 게임은 E에 해당하고 게임 속에서 체크메이트는 T에, P는 다음 게임에서 이길 확률로 볼 수 있다.

다른 예로, 이메일을 분류하는 스팸검사기가 있다고 할때

- *E*: Wathing you label emails as spam or not spam.
- *T*: Classifying emails as spam or not spam.
- *P*: The number(or fraction) of emails correctly classified as spam/not spam.

Supervised Learning

> **Given the rihgt answer** for each example in the data

즉, 주어진 정답이 있을때 사용할 수 있다. 이런 문제들은 많은데, *Regression* 이나 *Classification* 등이 있다.

- **Regression:** Predict continuous valued output
- **Classification:** Discrete valued output (0 or 1)

단순히 1개 혹은 2개의 attribute 를 사용할 수 있지만, infinite number of features (attribute) 를 사용하는 *Support Vector Machine* 같은 알고리즘도 있다.

Unsupervised Learning

즉 모든 데이터에 attribute 는 있지만 주어진 정답이 없을때 사용한다. 다시 말해서, 여러 집단으로 분류될때 미리 컴퓨터에게 이건 `type1` 이야 등의 정보를 제공하지 않는다.

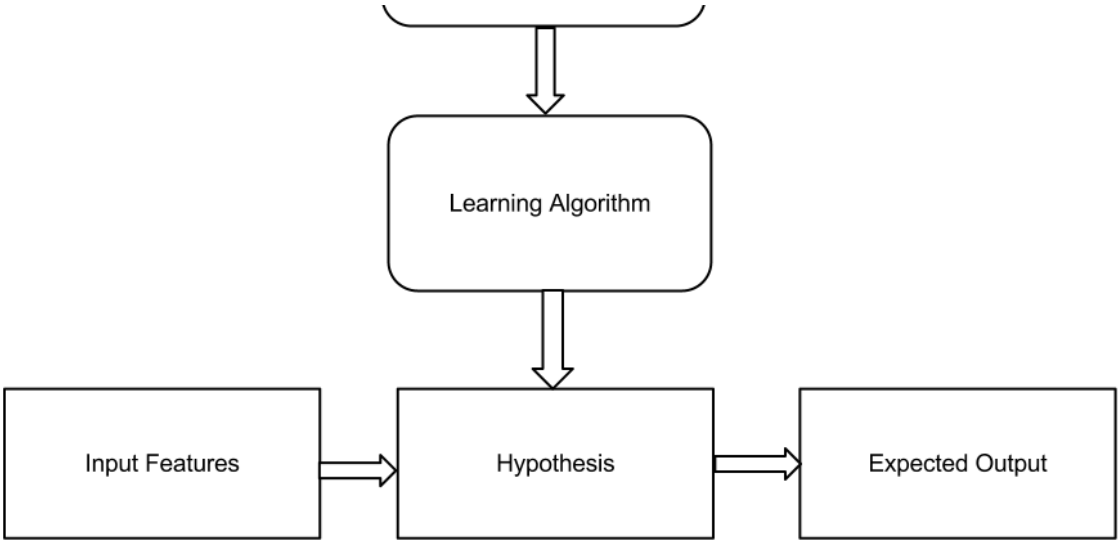
예를 들어서, 다음의 두가지 예는 *Unsupervised leanring* 이 아니라 *Supervised learning* 이다.

> (1) **Given email labeled as spam/not spam**, learn a spam filter
>
> (2) **Given a dataset of patients diagnosed as either having diabetes or not**, learn to classify new patients as having diabetes or not

Clustering 이라 불리는데, DNS Clustering, Social network analysis, market segmentation 등에 쓰인다.

Cocktail party problem 은 2명이 동시에 말하고, 이걸 서로 다른 위치에 있는 마이크가 녹음한다고 할 때 이 소리를 구분할 수 있는가 하는 문제다. 이것 또한 *Unsupervised lerning* 으로 해결할 수 있다.

Model Representation



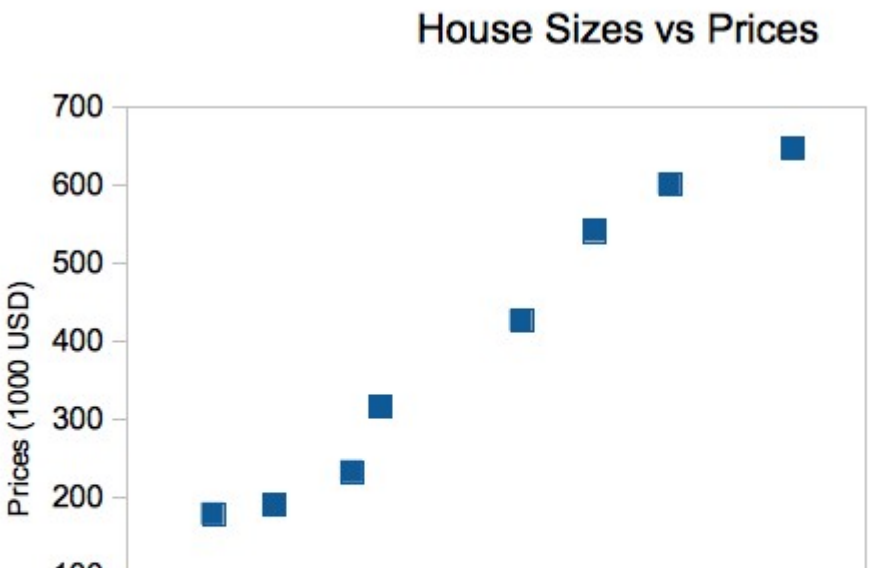
(http://mercris.files.wordpress.com/2012/07/genericmlatwork.png)

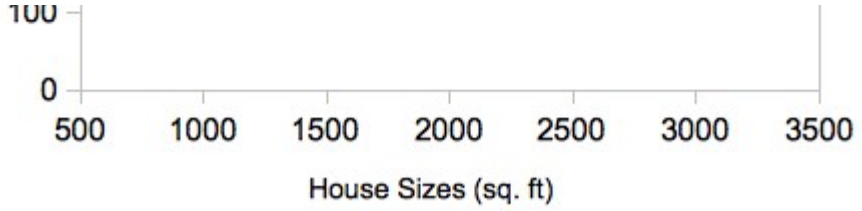
Traning Set 을 넣고 *Learning Algorithm* 을 돌리면 *Hypothesis* 가 나오는데, 이건 사실 함수라 보면 된다. 여기에 새로운 *Input X* 를 넣으면 *Estimated Y* 가 나온다.

참고로, 변수가 하나인 *Linear regression* 은 **Univariate linear regression** 이라 부른다.

Cost Function

예를 들어서 다음과 같은 데이터셋이 있을때,





http://mercris.files.wordpress.com/2012/07/screen-shot-2012-07-17-at-2-12-05-pm.png?w=584

H(hypothesis) 가 다음처럼 나온다면

$$h_\theta x = \theta_0 + \theta_1 x$$

여기서 `θ (Theta)` 는 *parameter* 라고 부른다. 문제는, 상수를 어떻게 찾느냐인데, 아이디어는 간단하다. training set `(x, y)` 에 가까운 `h(x)` 를 찾으면 된다.

따라서 다음과 같은 식을 만들 수 있고,

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

`J(01, 02)` 를 최소화 하는 `(01, 02)` 를 찾으면 된다. 이 식을 **cost function** 또는 **squred error function** 이라 부른다. 여기서 `1/2m` 으로 나누는 이유에 대해 좀 궁금해서 구글링 해봤는데, `1/m` 으로 나누는 이유는 *squared error* 에 대해 *mean* 을 얻기 위한거고, `1/2` 로 다시 나누는 이유는 미분했을때 나오는 `2` 를 제거하기 위해서다. SO 답변 을 첨부하면,

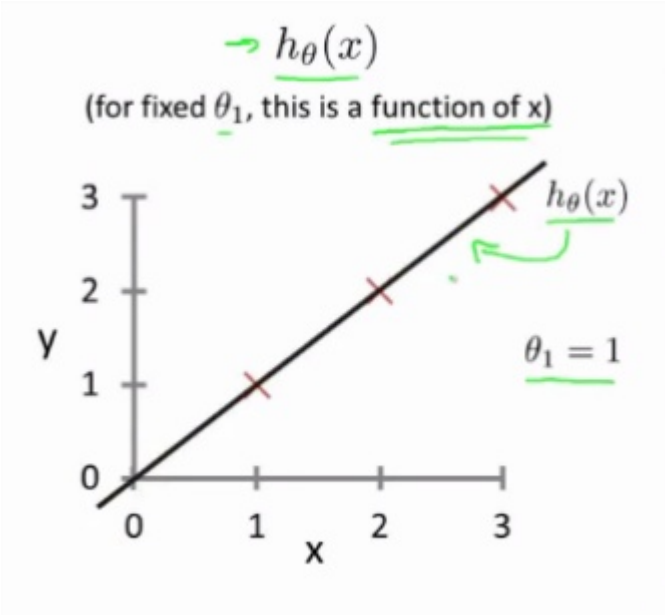
> The cost function is
>
> J(theta_0, theta*1) = 1/(2m) * sum*(i=1)^m [h_theta(x^i) - y^i]^2 By h_theta(x^i) we denote what model outputs for x^i, so h_theta(x^i) - y^i is its error (assuming, that y^i is a correct output).
>
> Now, we calculate the square of this error [h_theta(x^i) - y^i]^2 (which removes the sign, as this error could be both positive and negative) and sum it over all samples, and to **bound it somehow we**

> **normalize it - simply by dividing by m,** so we have mean (because we devide by number of samples) squared (because we square) error (because we compute an error):
>
> 1/m * sum_(i=1)^m [h_theta(x^i) - y^i]^2 **This 2 which appears in the front is used only for simplification of the derivative**, because when you will try to minimize it, you will use the steepest descent method, which is based on the derivative of this function. Derivative of a^2 is 2a, and our function is a square of something, so this 2 will cancel out. This is the only reason of its existance.

이 *cost function* 은 *regression* 문제를 위해 자주 쓰이는 기법이다.

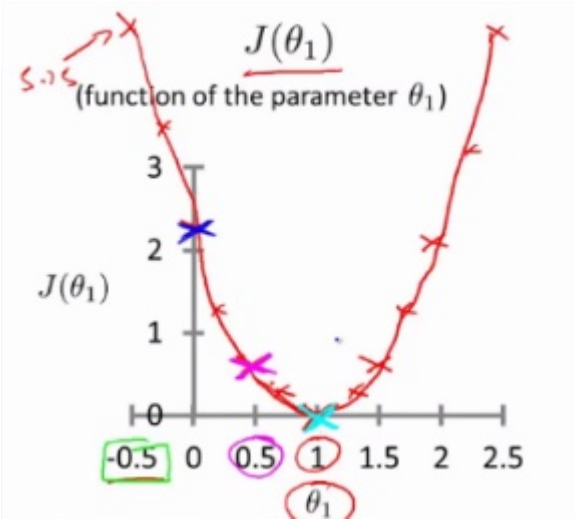
Cost Function: Intuition 1

Cost function 에서 만약에 `θ_0` 이 제로라면 `θ_1` 만 찾으면 된다. 따라서 다음과 같은 실제 데이터에서



(http://mapository.tistory.com/59)

`J(θ_1)` 을 찾아보면, 다음과 같은 이차함수가 나온다.

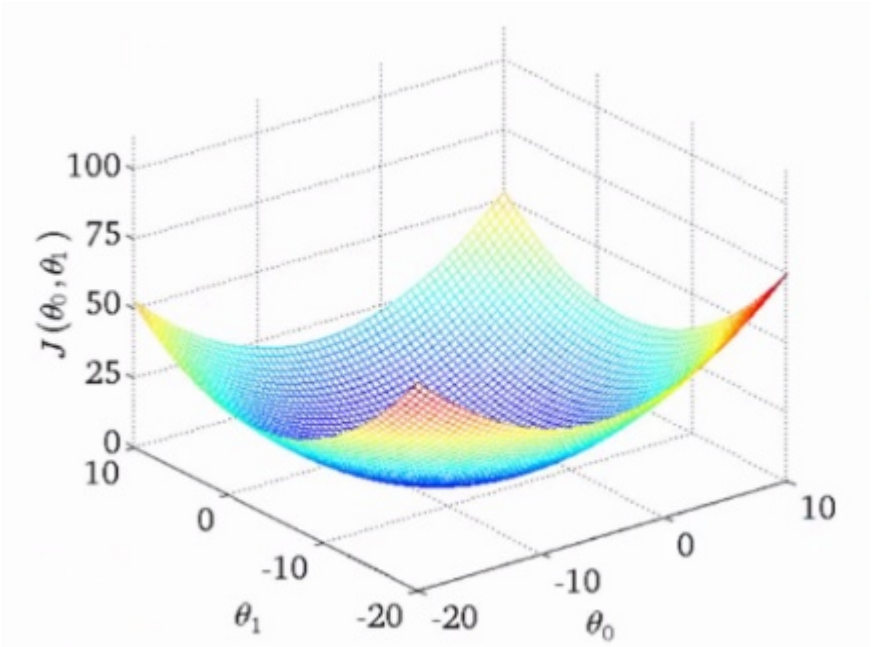


(http://mapository.tistory.com/59)

당연히 이차함수이므로, 기울기가 0이 되는 지점은 `J(0_1)` 을 미분해서 찾으면 된다. (이래서 아까 1/2가 있던 것)

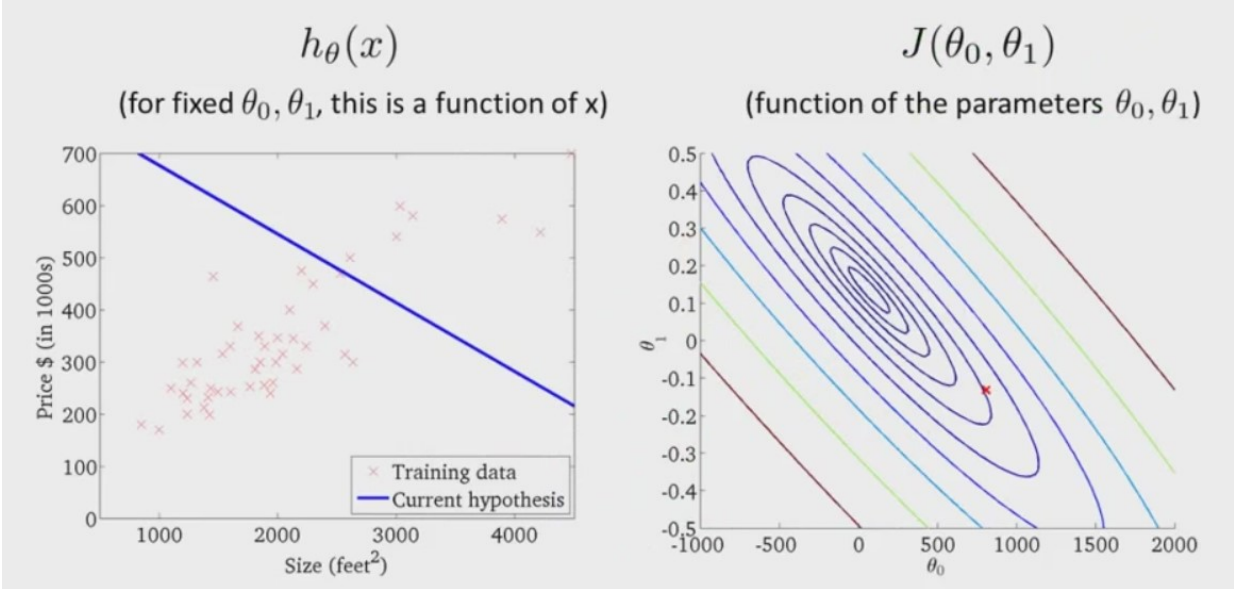
Cost Function: Intuition 2

Parameter 가 `0_1` 만 있었을때는, (`0_0` = 0) `J(0_1)` 이 이차함수였지만, `J(0_0, 0_1)` 일때는 다음과 같은 모양을 보여준다.



(http://mapository.tistory.com/59)

여기서 `J(0_0, 0_1)` 값을 제외하고 `(0_0, 0_1)` 을 평면으로 나타내면 아래 사진에서 우측과 같은 여러 궤도가 나온다.

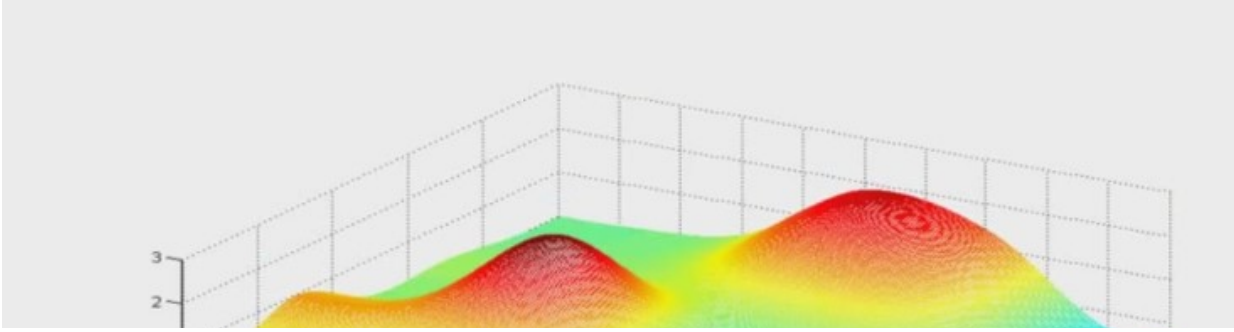


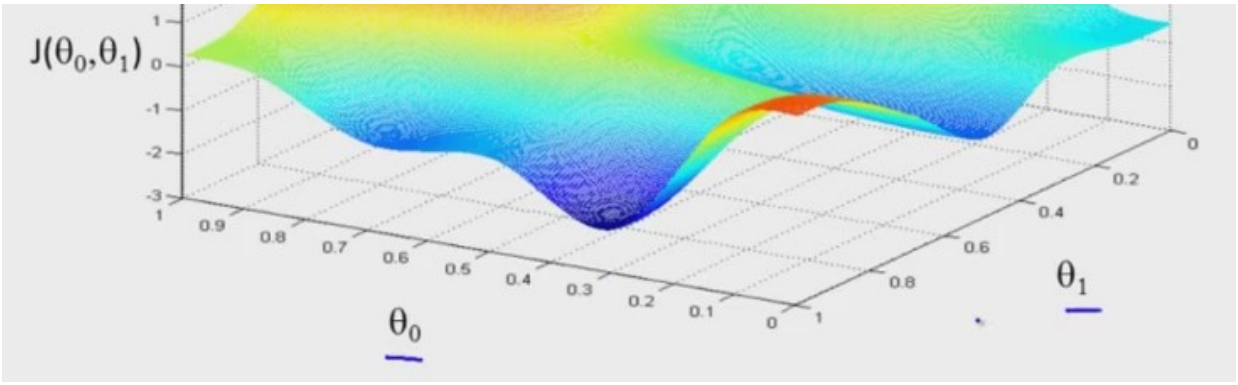
(http://mapository.tistory.com/59)

여기서 같은 궤도에 있는 `(0_0, 0_1)` 쌍은, 같은 `J` 함수를 만든다. 그리고 재밌는 사실은 궤도가 가장 좁은 타원의 중심에 있는 `(0_0, 0_1)` 가 가장 작은 `J(0_0, 0_1)` 를 만들어 낸다.

Gradient Descent

Gradient Descent 알고리즘은 *Linear Regression* 에만 쓸 수 있는건 아니고, 범용적인 알고리즘이다. *cost function* 의 최소값을 찾기 위해 사용할 수 있는데, 다음과 같은 `J` 가 있을때,





(http://mapository.tistory.com/59)

높이를 비교해 가며 점점 낮은쪽으로 이동해 가면서 `J` 의 최소값을 찾을 수 있다. 식은 다음과 같은데,

Gradient descent algorithm

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$
$$(\text{for } j = 1 \text{ and } j = 0)$$
$$\}$$

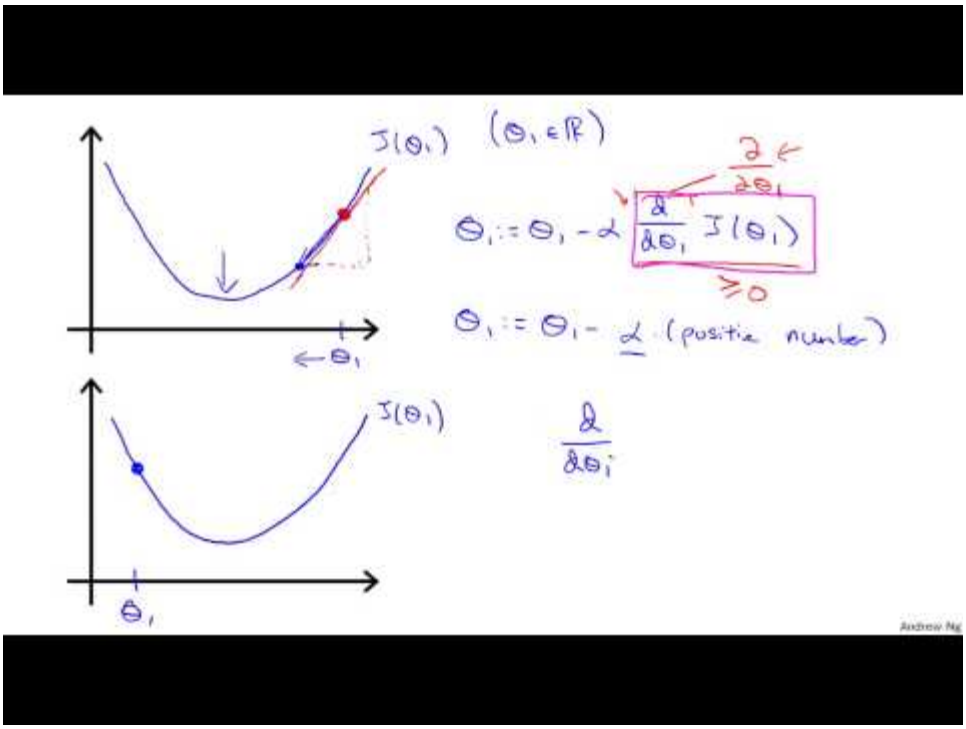
(http://econometricsense.blogspot.kr/2011_11_01_archive.html)

여기서 `:=` 는 *assignment* 다. `a(alpha)` 는 *learning rate* 라 부른다. 이때 `(0_0, 0_1)` 은 동시에 업데이트 되야한다. **(Simultaneous update)**

Gradient Descent: Intuition

이제 저 식을 분해하기 위해 `J(0_1)` 처럼 *parameter* 하나만 놓고 보면, 이차원 함수가 나올테다. 만약 현재 `0_1` 이 이차함수의 최저점 우측에 있다면, `J(0_1)` 을 미분한 값**(Slope, 기울기)** 에 양수 `a` 를 곱한 값을 `0_1` 에서 빼면서 갱신하면 `0_1` 은 점점 최저점 쪽으로 간다,

반대로 `0_1` 이 `J(0_1)` 의 좌측에 위치한다면 우측으로 이동하고, 아래는 그걸 요약한 그림이다.

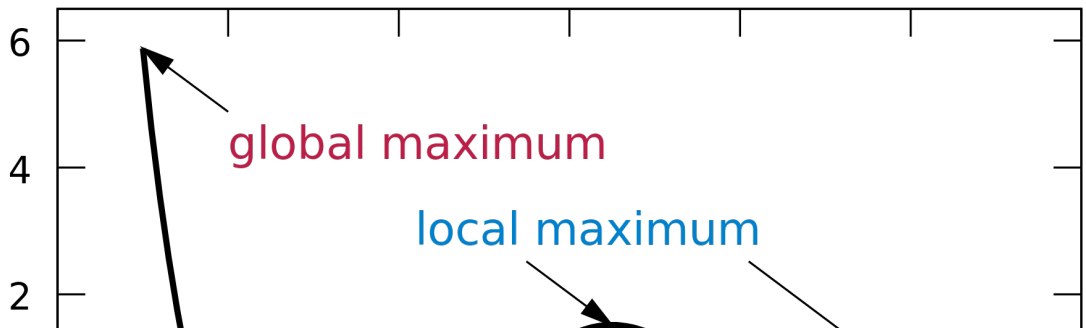


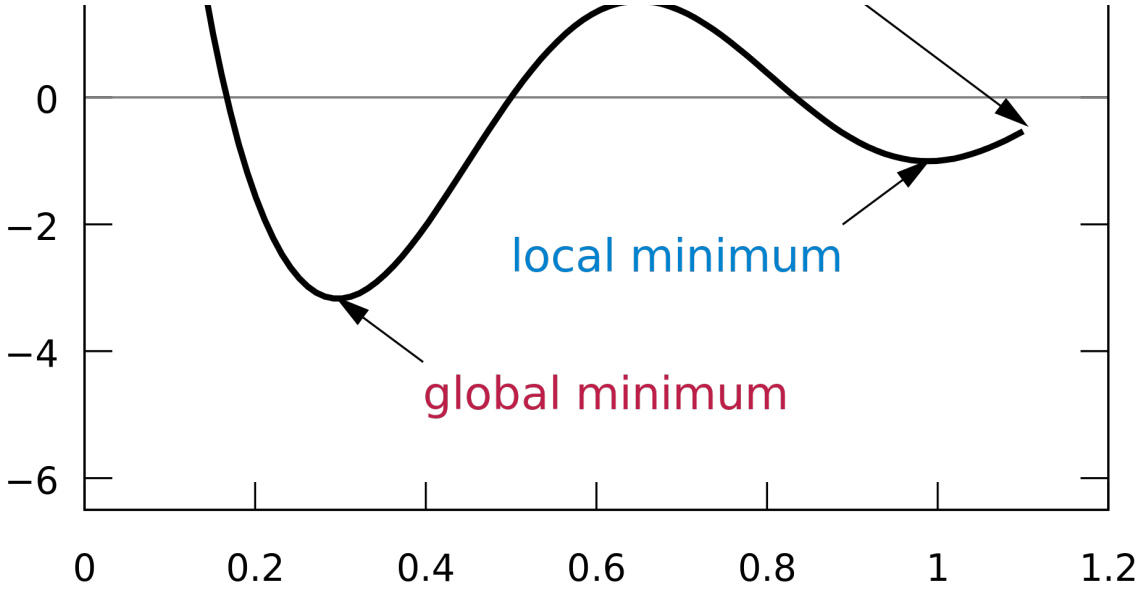
따라서 *learning late* `a` 가 너무 작으면 *Gradient descent* 가 너무 느려진다. 왜냐하면 `θ` 의 차이가 점점 작이지기 때문에 최저점에 도착할때 까지 너무 많은 step 이 필요하다.

반대로 너무 크면 최저점을 넘어갈 수 있다. 심지어 최저점에서 점점 더 멀어질 수 있다.

> if `a` is too small, *gradient descent* can be slow
>
> if `a` is too large, *gradient desscent* can overshoot the minimum, It may fail to converge, or even diverge

그런데 이 *gradient descent* 알고리즘의 문제는 **local optimum** 수 있다는 점이다. 왜냐하면 **local optimum** 에서도 `J` 의 derivative 가 `0` 이기 때문이다.





http://en.wikipedia.org/wiki/Backpropagation

Gradient Descent For Linear Regression

이제 *cost function* 을 *gradient descent* 에 집어넣고, 정리하자. `0_0(Theta zero)` , 과 `0_1(Theta one)` 대해서 시그마 내부 제곱을 각각 미분해서 정리하면,

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

(
http://pingax.com/linear-regression-with-r-step-by-step-implementation-part-1/)

참고로 **Convex function** 은 *Bowl shaped* 처럼 *local optima* 가 없는 `h`(Hypothesis) 를 말한다. 따라서 *convex function* 을 선택할 수 있다면, 하는편이 낫다.

Batch *gradient descent* 는 모든 training example 을 사용하는 *gradient descent* 를 말한다. (시그마에서)

어떤 경우에는 *gradient descent* 같은 interative algorithm 없이도
`min J(0_0, 0_1)` 를 풀 수 있다.

References

(1) http://mercris.wordpress.com/
(2) http://mapository.tistory.com/
(3) http://econometricsense.blogspot.kr
(4) http://pingax.com/